Misbehavior Forecasting for Focused Autonomous Driving Systems Testing

M M Abid Naziri*
mnaziri@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

Stefano Carlo Lambertenghi‡
Technical University of Munich, fortiss GmbH
Munich, Bayern, Germany

Andrea Stocco†‡
andrea.stocco@tum.de
Technical University of Munich, fortiss GmbH
Munich, Bayern, Germany

Marcelo d'Amorim*
mdamori@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

ABSTRACT

Simulation-based testing is the standard practice for assessing the reliability of self-driving cars' software before deployment. Existing bug-finding techniques are either unreliable or expensive. We build on the insight that near misses observed during simulations may point to potential failures. We propose Foresee, a technique that identifies near misses using a misbehavior forecaster that computes possible future states of the ego-vehicle under test. Foresee performs local fuzzing in the neighborhood of each candidate near miss to surface previously unknown failures. In our empirical study, we evaluate the effectiveness of different configurations of Foresee using several scenarios provided in the CARLA simulator on both end-to-end and modular self-driving systems and examine its complementarity with the state-of-the-art fuzzer DriveFuzz. Our results show that Foresee is both more effective and more efficient than the baselines. Foresee exposes 128.70% and 38.09% more failures than a random approach and a state-of-the-art failure predictor while being 2.49× and 1.42× faster, respectively. Moreover, when used in combination with DriveFuzz, Foresee enhances failure detection by up to 93.94%.

1 INTRODUCTION

Simulation-based testing [26, 34] is the de facto approach for Autonomous driving systems (ADS) testing [52, 65] as real-world physical testing, albeit important, has severe time, resource, and legal limitations [48]. Simulators enable developers to assess the reliability of the ADS before deployment as they consist of virtual simulation platforms in which developers "plug" their ADS and test it against challenging conditions. A test describes a route that an ADS must complete within a map representing an urban environment containing static and dynamic objects (e.g., traffic signs and other vehicles). Prior work has been proposed to generate test cases for ADS, particularly leveraging search-based optimization [1, 4, 5, 15, 34, 35, 37, 42] and fuzzing [9, 22, 28, 31, 64], which are characterized by drawbacks in terms of effectiveness and efficiency. Concerning the former, these solutions require the exploration of a vast multi-dimensional search space to pinpoint critical conditions. This is problematic due to the significant time and resource overhead, as running a single test case on a driving simulation platform can take several minutes. About the latter, existing testing techniques either apply mutations *at the scenario level* in the initial state of a simulation, e.g., the number and position of the vehicles at the beginning of the simulation, or guide the search relying on the *global* ADS behavior during the entire simulation or static features of test cases, such as the number of bends of a road [6]. As a consequence, these techniques are oblivious to *near misses*, i.e., circumstances where a failure would occur with minimal modifications to certain intermediate states of a simulation in which the ADS was close to failure. Overlooking near misses limits the potential of test generators to reveal failures and potentially inflates the safety perception of ADS.

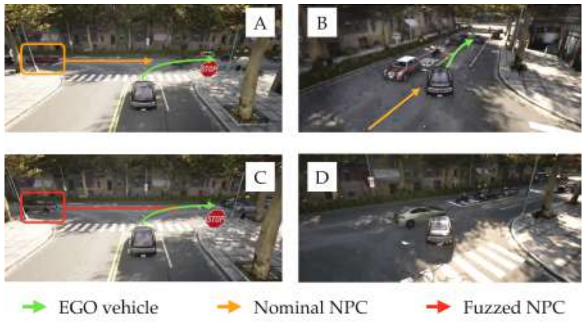

Figure 1: Illustrative example showing a near miss from a failure-free simulation (A/B) and collision from a sub-simulation which was mutated and clipped by Foresee (C/D).

This paper aims to improve simulation-based testing of ADS in terms of effectiveness and efficiency by investigating the problem of *detecting and exploiting* near misses during virtual simulations. Instead of relying on global or initial states of a scenario-based test, we target focused testing of intermediate states occurring during failure-free simulations. We leverage the insight that critical scenarios may exist within driving simulations, where even minor modifications, such as slight speed adjustments at an intersection, could expose failures.

We propose Foresee (FOREcasting unSafe Events and Emergency situations), a *focused* system testing technique for ADS. Foresee uses a monitor to measure risk during the simulation of a given test case. It uses risk data to derive, classify, and prioritize short-running test cases. Foresee fuzzes the inputs of these local and seemingly relevant test cases to find failures. More specifically, Foresee uses telemetry data (i.e., velocity, yaw rate, position, and relative distances to other agents) collected during simulation to automatically identify conditions in which the system is close to failure. This paper focuses on the identification of collisions, being the primary acceptance criteria for the safe deployment of ADS. We show that telemetry data offers clues about the failure likelihood of the ADS. Foresee forecasts potential failing conditions, such as the identification of vehicles or pedestrians that are crossing the future trajectory of the vehicle under test. Hazardous driving conditions are detected when the failure likelihood increases in a future state of the simulation, as predicted by Foresee.

Figure 1 shows an illustrative example of a near-miss situation and Foresee in action. The images **A** and **B** display consecutive snapshots of a nominal failure-free simulation. Note that the (future) trajectory of the ego vehicle and the trajectory of the NPC vehicle would eventually cross. However, due to the low speed of the NPC, the two vehicles do *not* collide (image **B**). We refer to such missed cases from the original simulations as “near misses”. Images **C** and **D** display a failure condition that Foresee reports. Foresee identifies the simulation segment at which the vehicles would have eventually intersected as a risky point. Then, it “clips” the risky segment from the original simulation and creates new focused simulations derived from the clipped segment. In this example, the new simulation is obtained by mutating the model of the NPC vehicle (image **C**). The new simulation results in a collision between the ego vehicle and the mutated NPC, which is faster than the original NPC (image **D**).

We evaluated the effectiveness of Foresee in the CARLA simulator [13], using ADS available from the literature and a diverse set of complex urban scenarios in which we observed many near-miss situations [54]. In our experiments on 34,416 simulations accounting for more than 609 individual failures, Foresee was able to surface up to 6.74% additional failures from near misses, a 38.09% increase with respect to SelfOracle [51], a state-of-the-art misbehavior predictor based on autoencoders, and a 128.70% increase with respect to a random assessment of risk. Foresee also demonstrates its efficiency against all the other baselines by discovering collisions 1.42× faster than SelfOracle and 2.49× faster than Random. We also observed a 318.29% more efficient discovery of failures with the best-performing configuration of Foresee relative to a ground truth generated from an exhaustive search. Additionally, we show that Foresee complements the state-of-the-art fuzzer DriveFuzz up to a 93.94% failure exposure increase.

Our paper makes the following contributions:

Technique. A technique for forecasting ADS misbehavior based on the kinematic state of the ego vehicle. Our approach is implemented in the publicly available tool Foresee [56].

Evaluation. An empirical study showing that Foresee’s risk-based assessment outperforms a random and a black-box approach SelfOracle [51] in terms of near-misses detection and improves the fuzzer DriveFuzz [28] in terms of failure exposure.

Dataset. A dataset of ADS failures to evaluate the performance of failure prediction systems and test generators for ADS. The tool and evaluation data are publicly available [56].

2 BACKGROUND

ADS are software systems developed with increasing capabilities to drive vehicles autonomously. ADS are usually designed following a modular architecture that includes perception, prediction, planning, and control modules [3, 25, 62]. Driving simulation platforms are the de facto choice in the industry for developing and testing ADS before real-world testing on roads [17, 18, 48]. High-fidelity simulators, such as CARLA [13] or LGSVL [43], replicate complex urban and highway driving scenarios using existing graphic assets representing buildings, pedestrians, and vehicles. These simulators generate a stream of simulated sensor data at regular intervals, such as data from cameras, that is fed to the ADS to enable motion within the simulated environment. The simulator relies on a model of physics that resembles real-world vehicle dynamics, allowing an accurate replication of vehicle behaviors and their corresponding trajectories. In the remainder of this section, we describe the nomenclature used in this paper.

Scenario. Scenarios define high-level traffic situations involving vehicle movements and interactions, commonly used for testing ADS behavior. These are often derived from real-world data, such as “pre-crash” reports from agencies like the US NHTSA [39], and are used to evaluate safety-critical responses. A scenario specifies the sequence of events in a simulation, including interactions with actors like pedestrians jaywalking or vehicles running red lights. It incorporates both static elements (e.g., traffic lights, crosswalks, trees) defined by the map layout, and dynamic elements such as the ego vehicle and Non-Playable Characters (NPCs), which follow scripted behaviors. Scenarios help uncover failures by capturing both routine and unexpected interactions between vehicles, infrastructure, and pedestrians.

Test Case. Given a scenario, a simulation-based test case is characterized by an *initial state* and a *route*. The initial state outlines the conditions of both static and dynamic objects at the beginning of the simulation, including the positions, velocities, and states of all objects in the scenario. The route specifies the path the ego vehicle is expected to follow during the simulation. It is typically defined in terms of a starting and ending point or as a sequence of waypoints within the map through which the ego vehicle should navigate. In summary, a route establishes a possible ground truth trajectory that the ego vehicle should follow in the simulation.

Failures. ADS are designed to meet several requirements, encompassing factors about passenger safety and comfort [8]. Driving simulation platforms automatically log any rule violations that occur during testing. Among these, safety violations are of utmost concern, particularly when it comes to autonomous driving, as they can potentially lead to vehicle crashes and casualties. This work focuses on predicting collisions. Collision avoidance is a primary prerequisite to be met as a self-driving vehicle must stay in its lane and prevent collisions to gain public trust and acceptance for production use. Our categories of failures include collisions involving the ego vehicle with elements beyond the road, such as pavements or poles, pedestrians, or other vehicles (Section 4.4).

3 APPROACH

Foresee aims to detect the occurrence of unexposed system failures during simulation-based testing of ADS. It builds on the observation that infractions are relatively rare compared to near misses. For example, on average, DriveFuzz [28] exposed 19 violations in 360 minutes (≈19 minutes per failure), whereas AV-Fuzzer [31] exposed,

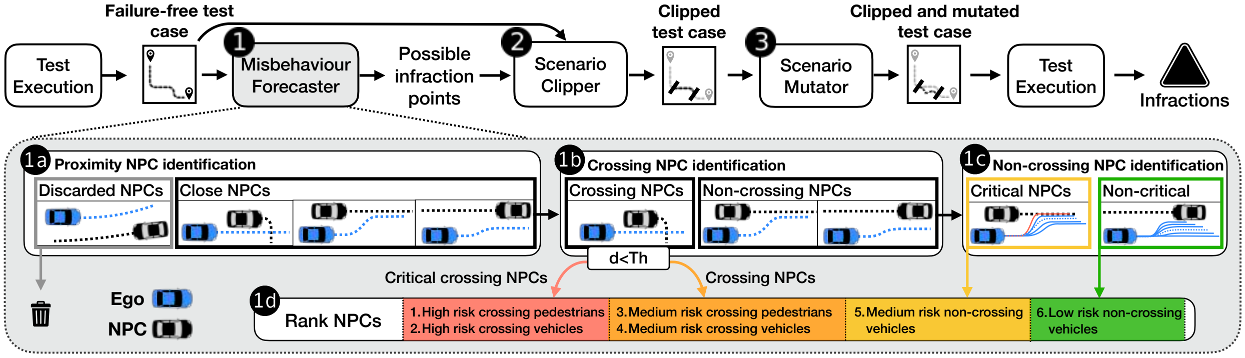
Test Execution
Failure-free test case
1
Misbehaviour Forecaster
Possible infraction points
2
Scenario Clipper
Clipped test case
3
Scenario Mutator
Clipped and mutated test case
Test Execution
Infractions
1a Proximity NPC identification
Discarded NPCs
Close NPCs
1b Crossing NPC identification
Crossing NPCs
Non-crossing NPCs
d<Th
1c Non-crossing NPC identification
Critical NPCs
Non-critical
Critical crossing NPCs
Crossing NPCs
Ego
NPC
1d
Rank NPCs
1.High risk crossing pedestrians
2.High risk crossing vehicles
3.Medium risk crossing pedestrians
4.Medium risk crossing vehicles
5.Medium risk non-crossing vehicles
6.Low risk non-crossing vehicles

Figure 2: An overview of Foresee (top), and the logic of the Misbehavior Forecaster in detail (bottom).

on average, 50 failures in 1,000 simulations (20 simulations per failure) [64]. The unique aspect of Foresee is that *it exploits near misses observed during simulations* to detect failures.

A test suite *TS* consists of test cases associated with scenarios that are challenging for an autonomous vehicle. For example, Scenario 4 of the CARLA leaderboard [54] deals with situations where the ego vehicle finds an obstacle on the road while performing a maneuver, and it must perform an emergency brake or an avoidance maneuver.

Foresee focuses on failure-free test cases, aiming to uncover missed failures that occur in near-critical conditions. This focus is motivated by the high cost of simulation-based testing, which makes it valuable to reuse simulations that already exhibit such near-critical behavior. Although any test could theoretically be forced into failure with unrealistic changes (e.g., excessive NPC speeds), Foresee prioritizes preserving realism. It introduces only small, targeted mutations and applies sanity checks to ensure consistency with the original scenario's intent.

3.1 Overview

Foresee takes as input a test case that does not reveal failures in nominal conditions (i.e., a failure-free test case) and reports infractions as output. The upper portion of Figure 2 illustrates the Foresee pipeline, consisting of three tasks, namely Misbehavior Forecaster, Scenario Clipper, and Scenario Mutator.

The Misbehavior Forecaster ❶ is responsible for identifying risky conditions during a simulation by tracking the position, speed, and steering angle of the NPCs at each time frame. Using the information in the collected execution traces, it then predicts which parts of the simulation are more likely to cause infractions. This component reports a ranked list of simulation timestamps sorted by a criticality score based on the probability of the ego vehicle to intersect NPC future trajectories; we hereafter refer to this list's elements as risky points. Foresee attempts to find potential modifications in the original simulation around these risky points to reveal previously unforeseen failures. To this aim, the Scenario Clipper ❷ is used to reconstruct a feasible CARLA-runnable scenario in the neighborhood of a risky point. For each risky point, Foresee computes start and end points from the original simulation that include the given likely infraction-inducing point. Then, it retrieves the list of NPCs relevant for that subset of the simulation. This is done to reduce unnecessary computation and focus risk assessment on meaningful threat sources. Finally, the Scenario Mutator ❸ introduces mutations in the initial states of the new scenario, effectively creating small variations on the intermediate states of the original simulation. At the end of the process, Foresee runs the derived (short) simulations and reports test cases revealing infractions.

In summary, Foresee uses a combination of misbehavior forecasting, scenario clipping, and local scenario-level mutation to find unforeseen failures. It reports failure-revealing test cases without executing expensive fuzzing campaigns.

3.2 Misbehavior Forecaster

In this paper, we propose a novel type of predictor in which risk is assessed using telemetry data to forecast potential failing conditions, such as vehicles or pedestrians that are crossing the future trajectory of the vehicle under test.

The misbehavior forecaster takes a test case *tc* as input and returns a list of risky points ranked by their likelihood of causing an infraction. The lower part of Figure 2 illustrates the four-step workflow of Foresee to obtain the ranked list of NPCs.

Step 1a: Proximity NPC identification. This step identifies the NPCs that approach the ego vehicle within a certain radius during the original simulation. For that, Foresee computes the set *Close NPCs* describing the circumstance when NPCs are closest to the ego vehicle, considering that radius. The set contains pairs with the simulation frame of the close encounter and the simulation ID of the corresponding NPC, and is further categorised in subsequent steps. Any NPCs that are not within the threshold radius are not considered (*Discarded NPCs*).

Step 1b: Crossing NPC identification. For each NPC identified in the previous step, Foresee retains those that cross the ego vehicle trajectory for further categorization. This filtering process yields two groups of NPCs: *Crossing NPCs* contains NPCs that intersect with the ego vehicle path during any simulation frame, and its subset, *Critical crossing NPCs*, includes NPCs that cross the ego vehicle but only within a limited distance from it. The remaining NPCs are identified as *Non crossing NPCs*= $\{x \mid x \in$ *Close NPCs* and $x \notin$ *Crossing NPCs*$\}$ and are further analyzed in step 1c.

Step 1c: Non-crossing NPC identification. Foresee uses this step to identify vehicle trajectories that currently do not cause collisions, but could result in collisions with the ego vehicle with minimal changes, such as slight modifications of NPC behaviour (e.g., speed, or steering profiles). For each NPC in set *Non crossing*

NPCs, Foresee generates N perturbations of the original ego vehicle trajectory by introducing a small error in the velocity and yaw rate values, ensuring that the perturbations remain within acceptable localization error bounds, following the existing guidelines [41].

It then evaluates if any of the newly generated ego vehicle trajectories is below a distance *threshold* to the NPC. If so, Foresee saves the NPC in the set *Critical NPCs*. All *Non-crossing NPCs* ids that do not meet this distance requirement are saved in *Non-critical NPCs*. The goal of this procedure is to account for the randomicity of the self-driving system under test, which could behave in a slightly different manner between different runs of the same scenario, therefore producing an unforeseen, dangerous situation.

Step 1d: Rank NPCs. In this step, Foresee ranks the risky points associated with the NPCs in the aforementioned groups according to their risk of causing a collision with the ego vehicle, and based on the type of NPC: pedestrians or vehicles (see step 1d of Figure 2). If multiple NPCs are associated with the same risk level, Foresee gives a higher score to NPCs that come closer to the ego vehicle trajectory during the simulation. For each of the ranked NPCs, our approach collects the simulation *frame* at which the actor comes closest to the ego vehicle during the nominal simulation.

3.3 Scenario Clipper

The scenario clipper component of Foresee is responsible for creating a scenario reflecting solely the risky conditions observed during the execution of the original scenario. We use the term “clip” to indicate that only a subset of the original scenario is retained.

The top n_{rp} risky points identified by the Misbehavior Forecaster (Section 3.2) determine the timestamps at which the simulation will be clipped, where n_{rp} is a hyperparameter that determines how many risky points should be considered for clipping. The parameters o_b and o_a indicate the length of the clip. For each chosen risky point *rp*, Foresee clips the scenario from timestamp $rp - o_b$ to timestamp $rp + o_a$. To restore the state of the original simulation at timestamp $rp - o_b$, Foresee saves the location, direction, and model of each NPC. In this way, the clipped scenario is “centered around” the risky point *rp*. For timestamp $rp - o_b$, our approach stores the exact location of the ego vehicle at that timestamp. This information is useful to set the starting waypoint s_{wp} for the clipped scenario. Concerning the ending waypoint e_{wp}, the selection is more challenging. Indeed, we observed that the location of the ego vehicle at timestamp $rp + o_a$ often results in invalid simulations in CARLA, because the simulator maintains specific sets of waypoints that can be used as a route in a scenario. As a working solution, our approach retrieves, from the log of the original simulation, a list of valid waypoints and uses the closest waypoint to the location of the ego vehicle at timestamp $rp + o_a$ as the ending waypoint e_{wp}. Finally, a new route XML file is created in which the clipped scenario starts at s_{wp} and ends at e_{wp}.

3.4 Scenario Mutator

For each clipped scenario, Foresee applies NPC-focused mutations. The rationale is to assess whether the ego vehicle can cope with situations that are *analogous* to the one observed during the riskiest parts of the original simulation, yet they are slightly different. For each risky point, Foresee generates c mutated children (Section 3.3), executes them, and reports the number of collisions. Our approach retains the original number of NPCs while varying certain properties. Two mutation operators are currently supported.

NPC Model Swapping. The Scenario Mutator swaps an NPC’s vehicle model with another of the same type. For example, a bicycle may be replaced with another bicycle or a pedestrian, while a car is only substituted with another car. This can impact the vehicle’s kinematic characteristics, such as speed differences between the original and replacement car. The replacements are limited to the “relevant” NPCs in the neighborhood of the ego vehicle. To identify these relevant NPCs, the misbehavior forecaster reports a ranked list of NPCs that approached the ego vehicle along with the time frame in which they were closest. Of these, only the closest is selected for mutation. Foresee uses this information to select the NPCs for which their time frame lies within the interval $[rp - o_b, rp + o_a]$, where rp denotes the riskiest point selected from the ranked list and o_b and o_a denote, respectively, the offsets before and after the risky point delimiting the period of a new simulation.

Steering Angle Perturbation. The Scenario Mutator perturbs the current steering angle of the closest NPC to the ego vehicle as another form of mutation. To introduce variations in the steering angle, Foresee tracks the closest vehicle to the ego vehicle $npc_{closest}$ in a test case at the beginning of the risky interval, i.e., at $rp - o_b$. Afterwards, for each child simulation, the Scenario Mutator applies a random steering angle to $npc_{closest}$. The simulator accepts scalar values within the range of $[-1.0, 1.0]$; a value within this range is randomly selected for the steering angle, following thresholds suggested by Reid et al. [41].

Validity/Realism Check. To maintain the validity and realism of the original simulation, the Scenario Mutator produces new short-lived simulations introducing modifications in existing NPCs within the domain model and constraints of the CARLA simulator. This ensures that the resulting mutations are valid and realistic by design, as they operate within the NPC and kinematic space allowed in the CARLA simulator. The Scenario Mutator also ensures that the newly mutated vehicle model avoids collisions with other NPC vehicles upon spawning at the beginning of the simulation, potentially due to the increased length of the new model vehicle. To achieve this, our approach computes the distances between each pair of NPC vehicles and retains only the valid vehicle models. A valid vehicle model fits within the gap between two adjacent NPCs and does not cause immediate collisions. Additionally, certain sensors on the map at the initial point of a simulation can be placed as invisible objects, causing collisions with NPC vehicles if placed directly on the road. To avoid inflating the number of collisions with these phantom objects, we increment the z-axis value of the locations by a small constant value $z\_offset = 2$ when saving the location. Since the simulator accounts for gravity, these vehicles are automatically positioned on the ground upon spawning.

4 EVALUATION

4.1 Research Questions

RQ$_1$ (effectiveness): *How effective is Foresee in exposing misbehaviors in near-miss scenarios compared to exhaustive search? How does effectiveness vary with clip sizes and number of child tests?*

RQ$_2$ (comparison): *How does Foresee compare with alternative misbehavior prediction techniques (Exhaustive, Random and SelfOracle) and proximity-based ranking?*
RQ$_3$ (efficiency): *How efficient is Foresee in exposing misbehaviors in near-critical situations?*
RQ$_4$ (complementarity): *Does Foresee improve an existing state-of-the-art fuzzer?*
RQ$_5$ (generalizability): *Does Foresee generalize to other scenarios and industry-grade ADS?*

The first research question evaluates the ability of Foresee to detect near misses and generate failures. We evaluate effectiveness when varying two important parameters: the size of the clipped scenarios (as per o_a+o_b) and the number of near misses to exploit in each test (as per n_{rp}). Intuitively, longer clips and a higher number of risky points may reduce the tool's efficacy as the technique is tailored for targeted risk selection. We also compare the precision of Foresee against an impractical Exhaustive approach that approximates an upper bound on the possible number of failures by applying fuzzing at each timestamp (excluding the first o_b seconds).

The goal of the second research question is to measure the ability of Foresee's misbehavior forecaster component (Section 3.2) to detect risky points. We compare Foresee against two baseline approaches, namely Random and SelfOracle. The Random baseline selects a waypoint for local fuzzing at random, clips the simulation around that waypoint, and mutates its corresponding initial state. The second approach replaces the kinetics-driven misbehavior forecaster of Foresee with SelfOracle [51], a data-driven misbehavior predictor for ADS based on autoencoders.

The third research question evaluates how fast Foresee exposes misbehaviors. We hypothesize that Foresee exposes failures faster because only parts of the original simulation are used and executed. This research question evaluates this hypothesis.

The fourth research question evaluates whether Foresee can complement an existing state-of-the-art fuzzer, specifically DriveFuzz [28], by uncovering failures in non-failing test cases.

Lastly, the fifth research question evaluates the generalizability of Foresee across additional complex urban scenarios, and an evaluation using an industrial-grade ADS.

4.2 Objects: Simulator, Scenarios, and ADS

4.2.1 Simulator. We used the CARLA simulator for self-driving cars [13] (v. 0.9.10.1), a driving simulator developed with the Unreal Engine 4 [14] used in previous ADS testing literature [28, 35, 64]. We chose CARLA as it supports complex urban scenarios with many configurations of static and dynamic objects, and provides a rich set of sensors (e.g., cameras, LiDAR, GPS, and radar) to enable the observation of the status of the ADS throughout the simulation.

4.2.2 Scenarios. CARLA provides multiple closed-loop urban maps for ADS testing. We use Town10, a default map with standard environmental settings (e.g., sunny weather). Each map includes driving scenarios, defined by multiple test cases (called routes in CARLA). Table 1 summarizes the five selected scenarios, chosen for their diversity, totaling 120 test cases.

Table 1: Characterization of scenarios of Town10.

Scenario	Description	# Routes
3	NPCs cross in front of the ego vehicle, requiring braking	16
4	Obstacles appear post-turn, requiring reactive avoidance	46
7	NPCs run red lights; ego vehicle must avoid collisions	19
8	ego vehicle makes unprotected left turn, yields to traffic	19
9	ego vehicle turns right, yielding to crossing traffic	20

4.2.3 ADS under Test. We use InterFuser [44] and TransFuser [10], two ADS that achieve top performance on the CARLA leaderboard [53] and have been adopted in recent work [21, 23, 24].

InterFuser is a multi-modal fusion model designed for complex driving scenarios. It uses camera and LiDAR inputs, along with interpretable intermediate features (e.g., planned trajectory, traffic signals), to produce safe driving commands (e.g., steering, throttle). Features are extracted using ResNet backbones and fused via a transformer module, followed by a Gated Recurrent Unit for trajectory prediction. A safety controller uses high-level cues to constrain final control outputs.

TransFuser is a transformer-based fusion model that integrates spatial and temporal features from camera and LiDAR inputs using cross-attention [58]. Unlike InterFuser, TransFuser performs sensor fusion at a later stage, combining image and LiDAR features extracted via ResNet encoders. A waypoint prediction module outputs the ego vehicle vehicle's future trajectory, which informs low-level driving actions.

4.3 Competing Methods

Since direct baselines for our approach are not available we conduct experiments exploring different methods for detecting risky points and applying alternative ranking strategies, as described below.

4.3.1 Risk Scenario Identification. To evaluate risk-scenario identification, we compare Foresee against three baseline variants that retain the Scenario Clipper (Section 3.3) and Scenario Mutator (Section 3.4) components, but substitute the misbehavior forecaster with alternative strategies, described below.
Exhaustive. We compute an upper bound on failure detection by exhaustively sampling the simulation timeline, independent of risk scores. Although computationally expensive and impractical in real-world use, it serves as a reference for maximum possible coverage. Exhaustive selects every second of the original simulation as the center of a clip, ignoring risk scores, and constructs mutated sub-simulations from these segments. For instance, with $o_b = o_a = 3$ (i.e., a 6s clip), a 48s simulation yields 45 clips (skipping the first o_b seconds). Although these clips often overlap, creating significant computational overhead, mutations at different start times can still produce varied outcomes. This baseline illustrates how close a technique comes to uncovering all potential failures, serving as a proxy for the ground truth in near-miss detection.
Random. This approach randomly selects waypoints from the original route and uses them as focal points for clipping and fuzzing. For this baseline, instead of using the misbehavior forecaster, we randomly select waypoints from the original route and apply clipping and mutation around them. This comparison aims to show how Foresee compares against a technique that does not use any guidance to select segments for local fuzzing.

SelfOracle. SelfOracle [51] is a black-box ADS misbehavior predictor [51]. Even if SelfOracle was not proposed for test generation, this baseline is relevant because SelfOracle is designed to detect risky situations that result in failures of ADS. SelfOracle requires images captured by the front-facing camera for training and inference. We use the best configuration of SelfOracle presented in the original paper, i.e., a variational autoencoder (VAE) that reconstructs driving images and uses the reconstruction loss as a measure of confidence. For training SelfOracle, we collected 151 images at 20 FPS from the map of "Town10" since this map is used in all of our experiments. As the original training sets from the InterFuser and Transfuser papers are not available, we collected a training set size in line with what was described in the paper, i.e., 125k for "Town10" at 2 FPS. The autoencoder uses the Adam optimizer [29] to minimize the mean squared error (MSE) loss over 10 epochs, using a learning rate of 0.001. During inference, for each frame of the simulation, SelfOracle computes a reconstruction error; we average the reconstruction errors within pairs of consecutive waypoints (i.e., a segment). This average value indicates the risk of the segment, and we rank simulation segments by this risk value. We set a threshold $\gamma = 0.95$ for the expected false alarm rate in nominal conditions to identify risky conditions.

4.3.2 NPC Ranking. To evaluate the effectiveness of risk-scenario prioritization, we compare the risk-based strategy of Foresee's misbehavior forecaster with a proximity ranking method. Instead of performing the complete trajectory-based critical frame ranking used in Foresee (Section 3.2), we execute only the first step: Proximity NPC identification. This step consists of identifying all NPCs that are within a set radius during the original simulation. Thus, all NPCs in the *Close NPCs* set of Foresee's misbehavior forecaster are candidate risky points, which we then rank according to their order of appearance in the simulation.

4.4 Experimental Setup

4.4.1 RQ₁. We execute various configurations on the scenarios from Table 1 and their corresponding test cases. To identify NPCs in the proximity of the ego vehicle, we used the thresholds $th_1 = 10m$ and $th_2 = 50m$ for vehicles and pedestrians, respectively. For non-crossing NPC identification, we use the $threshold = 2m$. These values were tuned during preliminary pilot experiments.

For each near miss, we clipped the test case to contain the near miss within the offsets o_b and o_a. For simplicity, we considered offsets of the same size $o_b = o_a$. For each clipped test case, we generate $c = 4$ mutated test cases by injecting mutations according to Section 3.4 and execute the clipped and mutated test cases. We determine the number of risky points to analyze as follows. For each simulation, we identify risky points and consider the top n_{rp} risky points from the ranked list that the misbehavior forecaster (Section 3.2) reports, unless the number of risky points identified is less than n_{rp}, in which case we consider all identified risky points. We obtain the number of risky points n_{rp} from the set $\{1, 2, 4\}$. Considering the length of the generated test cases, we consider $o_b = o_a \in \{3, 5\}$, thus enabling sub-simulations 6s and 10s long. Regarding the ranking method, we compare proximity ranking, introduced in Section 4.3.2, with our Misbehavior Forecaster, which is based on risk likelihood. For Exhaustive, we extract *all possible* 6s and 10s clips from the original test cases by treating each second of the simulations as a risky point, i.e., taking $o_a + o_b$ seconds long cuts around each second of the original simulation. Given the high cost of this approach, we executed it only for InterFuser.

We use the number of collisions as the key performance metric for this research question (Section 2). We apply mutation operators on these clips, run the experiments with the same mutation count (c), and compute the number of failures for comparison.

Overall, our experiment evaluates 12 configurations of Foresee (i.e., 3 risky points × 2 simulation lengths × 2 ranking methods). We empirically observed that shorter times are likely to produce invalid simulations, whereas longer times jeopardize the benefits of local fuzzing and simulation reuse.

4.4.2 RQ₂. We execute Random and SelfOracle with the same experimental setup from RQ$_1$. More precisely, we use the same set of scenarios and tests (Table 1), the same set of mutation operators (Section 3.4), and the same configurations of Foresee (i.e., combinations of clip size and offsets). To assess the effectiveness of Foresee's risk-based strategy, we re-run it using the most effective clip size identified in RQ$_1$, but with only the first step of ranking—Proximity NPC identification—as described in Section 4.3.2.

4.4.3 RQ₃. We log the time that each failure was observed and report the failure rate over time as an area under the curve (AUC). In this case, the x-axis of the curve indicates time, and the y-axis of the curve indicates the cumulative number of failures observed. Intuitively, the larger the area, the better the efficiency.

To cope with the non-determinism of the driving platform, we executed all the experiments 3 times and reported averages. Our dataset has a total of 120 routes from 5 scenarios, 17 of which have infractions when executed with InterFuser and 9 have infractions when executed with Transfuser. Consequently, we discarded those cases, leaving 103 routes (i.e., test cases) for InterFuser and 111 for Transfuser. Each of these test cases is considered a seed scenario, and from each of these test cases, we derive a maximum of 4 risky points. We aimed to construct 4 clipped routes to get data for $n\_rp \in \{1, 2, 4\}$, as well as 4 mutated routes per clipped route from this collection of seed scenarios. Some seed scenarios do not have 4 risky points, so the number of clipped scenarios can be any value within the range $0 \le n_{rp} \le 4$. For InterFuser, we got a total of 297 clipped scenarios, and for Transfuser we got 181 clipped scenarios, which yields a total of 1,782 test cases for InterFuser and 1,086 test cases for Transfuser across 2 configurations ($(o_b + o_a) \in \{6s, 10s\}$) and 3 repetitions. Considering all configurations, we executed 34,416 test cases (3 techniques × 4 mutations × 1,782 routes for InterFuser and 3 techniques × 4 mutations × 1,086 routes for Transfuser).

It is worth noting that although test cases derived from the original input test are designed to be short running (i.e., 6s and 10s long), in practice, they tend to take longer than the estimated time because of traffic signals and vehicles getting stuck during the simulation. In our setting, our simulations took on average 15s for 6s-clips and 30s for 10s-clips. Thus, the average simulation time is 22.5 seconds, with the estimated total computing time of our experiments being more than 215.1 hours ($22.5 * 34{,}416/3,600$) or around 9 days. For the Exhaustive search, we created clipped and mutated sub-simulations from each second of each seed scenario (we omit

Table 2: RQ$_1$: Effectiveness of Foresee. Breakdown of collisions per scenario, configuration, and ADS. E=Exhaustive. #RPs=Number of risky points. #C.=Number of collisions.

	clip duration = 6s							clip duration = 10s						
n_{rp}	1		2		4		E	1		2		4		E
	#RPs	#C.	#RPs	#C.	#RPs	#C.	#C.	#RPs	#C.	#RPs	#C.	#RPs	#C.	#C.
InterFuser														
Scenario3	15	3	30	6	59	13	51	15	6	30	8	59	13	35
Scenario4	36	3	68	13	97	31	128	35	11	66	26	95	39	78
Scenario7	12	7	26	14	40	16	66	11	1	25	7	41	22	68
Scenario8	14	0	30	5	44	7	18	13	2	28	12	42	14	46
Scenario9	14	3	25	3	35	5	7	14	4	25	4	37	5	7
Σ	91	16	179	41	275	73	271	88	24	174	57	274	94	233
TransFuser														
Scenario3	13	1	25	2	45	2	-	15	7	29	11	51	17	-
Scenario4	28	2	47	3	64	6	-	29	4	47	9	64	9	-
Scenario7	11	0	15	0	17	0	-	12	2	16	2	18	2	-
Scenario8	13	0	18	0	22	0	-	12	0	17	0	21	0	-
Scenario9	13	0	23	0	25	0	-	13	0	23	0	25	0	-
Σ	78	3	128	5	173	8	-	81	13	132	22	179	28	-
Total	169	19	307	46	448	81	-	169	37	306	79	453	122	-

the first o_b seconds for each seed scenario, as it is not possible to obtain a valid simulation). In total, this process took around 43 days of execution time across 3 repetitions with InterFuser.

4.4.4 RQ$_4$. DriveFuzz is a feedback-directed test generator. We execute one iteration of DriveFuzz on the same 103 and 111 seed routes identified in RQ$_3$ for InterFuser and TransFuser respectively. We then collect the identified failing cases and conduct a second iteration of DriveFuzz on the remaining non-failing cases. The obtained InterFuser and TransFuser routes in which DriveFuzz did not detect any failures are then used to apply the best performing configuration of Foresee, identified in RQ$_1$. To answer this research question, a total of 120 full-scenarios and 480 10s-clips have been executed for each SUT, resulting in an estimated execution time of 16 hours.

4.4.5 RQ$_5$. We evaluate the ability of Foresee to generalize to varying maps, agents, and weather conditions. Specifically, we executed the best Foresee configuration on all scenarios of four additional maps, namely Town01, Town02, Town03, and Town04, for both the InterFuser and TransFuser agents. Moreover, we evaluate the best-performing configuration in Town10 under an adverse weather scenario characterized by high cloudiness, fog density, wetness, and precipitation, with the sun positioned at a 45-degree angle. This combination introduces reduced visibility and slippery road conditions, testing the agent's ability to maintain control and perceive the environment through both camera and LiDAR sensors. To avoid confounding effects, we avoid adding further mutations (e.g., model swapping or steering angle) to the sub-simulations and measure the impact of weather on the agents. Finally, to assess the applicability of Foresee to an industrial-grade autonomous driving system, we evaluate its capability to identify risky scenarios in the Apollo 8 framework [3].

4.5 Results

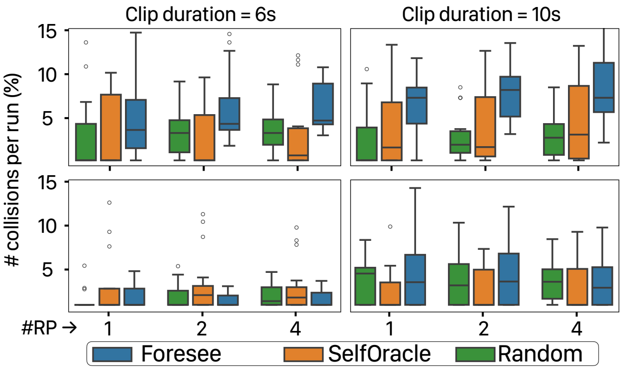

Figure 3: RQ$_2$: Comparison of different configurations.

4.5.1 RQ$_1$ (effectiveness). Table 2 details the results, reporting, for different clip durations (6s or 10s), the number of risky points (#RP), and collisions (#Colls.) for different numbers of risky points (1, 2, or 4) and for the two evaluated SUTs (InterFuser above, TransFuser below). For comparison, the column "#Colls." under Exhaustive represents the maximum number of collisions detected in an exhaustive search (Section 4.3.1), only for InterFuser. Intuitively, as the number of risky points increases, Foresee exposes more failures over time. For clip duration = 6s, the failure rate on InterFuser is 17.95% (16.33/91) when one risky point is selected, 22.86% (41/179.33) when two risky points are selected, and 26.51% (73/275.33) when four risky points are selected. On TransFuser, the rates are much lower, but maintain the same trend: 3.8% (3/78), 3.9% (5/128), and 4.6% (8/173), respectively. Similar considerations hold for the clip duration=10s on InterFuser where we observe failure rates of 27.27% (24/88), 32.76% (57/174), 34.19% (93.67/274) for the configurations with 1, 2, and 4 risky points selected, respectively. TransFuser achieves more modest gains: 16.0% (13/81), 16.7% (22/132), and 15.6% (28/179), suggesting diminishing returns as more points are fuzzed. The Exhaustive column demonstrates the number of collisions discovered by the exhaustive search and acts as the ground truth for the potential collisions discoverable by Foresee. For $o_b + o_a = 6$ and $n_{rp} = 4$, Foresee discovers 73 collisions compared to 271 collisions discovered by Exhaustive, demonstrating 26.94% coverage. Similarly, for $o_b + o_a = 10$ and $n_{rp} = 4$, coverage for Foresee is 40.14% (93.67x 100 / 233.33), demonstrating the best coverage amongst the 12 configurations of Foresee.

> **RQ$_1$ (effectiveness)**: *Foresee exposes many collisions from near misses, with failure rates ranging between 17.95-34.19% for InterFuser and between 15.6-16.7 for TransFuser.* We find the configuration with clip duration=10s and number of risky points=4 to provide the best trade-off in terms of failure rate.

4.5.2 RQ$_2$ (comparison). Figure 3 shows the distributions of the number of collisions for Foresee, SelfOracle, and Random with InterFuser (top) and TransFuser (bottom). The left and right figures show the results for clip durations of 6s and 10s, respectively. Foresee outperforms the baselines in all configurations except clip duration=6s for TransFuser where no technique achieves adequate performance. We measured the statistical significance of the differences using the nonparametric Wilcoxon rank-sum test [59], with $\alpha = 0.05$, and the magnitude of the differences using Cohen's effect size d [11]. To account for multiple comparisons,

Table 3: RQ$_2$: Comparison of Random, SelfOracle, and Foresee. Configuration 10 x 4 ($o_b + o_a = 10, n_{rp} = 4$).

		Random						SelfOracle						Foresee					
	#TCs	F1	F2	F3	F4	F5	Σ	F1	F2	F3	F4	F5	Σ	F1	F2	F3	F4	F5	Σ
InterFuser																			
Scenario3	235	0	5.00	0	2.67	0	7.67	0	2.00	0	1.67	0	3.67	0	9.33	0	4.00	0	**13.33**
Scenario4	390	0.67	0	0	10.00	3.67	14.33	0	0	0	8.33	31.67	**40.00**	0.67	0	0.67	26.00	11.67	39.00
Scenario7	167	0	0	0	4.67	0	4.67	0	0	0	18.67	0	18.67	0	0	1.67	20.00	0	**21.67**
Scenario8	179	0.33	0	0.33	4.67	0.33	5.67	0	0	0	1.67	0.67	2.33	0	0	0	14.33	0	**14.33**
Scenario9	145	0	0	0	0.33	0	0.33	0	0	0	0.33	0	0.33	0	0	0	5.33	0	**5.33**
Σ	1116	1.00	5.00	0.33	22.34	4.00	32.67	0	2.00	0	30.67	32.34	65.00	0.67	9.33	2.34	69.66	11.67	**93.66**
TransFuser																			
Scenario3	202	0	5.67	0	0	0	5.67	0	13.67	0	0	0	13.67	0	7.00	0	9.67	0.33	**17.00**
Scenario4	252	0	0	0	6.67	2.67	**10.00**	0	0	0	8.33	1.33	9.67	0.33	0	0	8.33	0.33	9.00
Scenario7	71	0	0	0	1.33	0	1.33	0	0	0	0	0	0	0	0	0	2.00	0	**2.00**
Scenario8	78	0.33	0	0	3.33	0	**3.67**	0	0	0	0	0	0	0	0	0	0.33	0	0.33
Scenario9	92	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Σ	695	0.33	5.67	0	11.33	2.67	20.67	0	13.67	0	8.33	1.33	23.34	0.33	7.00	0	20.33	0.66	**28.33**
Total	1811	1.33	10.67	0.33	33.67	6.67	53.34	0	15.67	0	39.00	33.67	88.34	1.00	25.33	2.34	89.99	12.33	121.99

Table 4: RQ$_2$: Statistical tests.

Configuration	p-Value	Corrected p-Value	Cohen's d	Effect Size
InterFuser: Foresee vs SelfOracle				
10 x 1	0.187622	0.375244	0.594433	medium
10 x 2	0.018066	0.126462	0.924394	large
10 x 4	0.000305	0.003355	0.781236	medium
6 x 1	0.346150	0.375244	0.525022	medium
6 x 2	0.055359	0.221436	0.910631	large
6 x 4	0.015076	0.120608	0.852315	large
InterFuser: Foresee vs Random				
10 x 1	0.041260	0.20629	1.017734	large
10 x 2	0.000854	0.00854	1.698214	large
10 x 4	0.000061	0.000732	1.669225	large
6 x 1	0.084260	0.25278	0.544179	medium
6 x 2	0.018538	0.126462	0.765743	medium
6 x 4	0.001160	0.01044	1.191313	large
TransFuser: Foresee vs SelfOracle				
10 x 1	0.028314	0.311454	0.690996	medium
10 x 2	0.010793	0.129516	0.505178	medium
10 x 4	0.028417	0.311454	0.297522	small
6 x 1	0.074735	0.597880	0.463355	small
6 x 2	0.035466	0.319194	0.64913	medium
6 x 4	0.123485	0.864395	0.584365	medium
TransFuser: Foresee vs Random				
10 x 1	0.344659	1	0.334029	small
10 x 2	0.479500	1	0.218747	small
10 x 4	0.813945	1	0.110475	negligible
6 x 1	0.446060	1	0.258276	small
6 x 2	0.498962	1	0.273033	small
6 x 4	0.326989	1	0.355722	small

we also applied Holm–Bonferroni correction [2] to our p-values. Table 4 shows the results of the statistical tests.

For most configurations with InterFuser, the differences between Foresee and both baselines are statistically significant (6x2, 6x4, 10x2, and 10x4), i.e., the p value < 0.05 with a medium to large effect size. For some configurations (6x1 and 10x1), only the differences between Foresee and Random are statistically significant, with medium/large effect sizes. The effectiveness of Foresee seems to be autopilot-dependent, as expected. InterFuser is a more stable agent, and configurations with richer sampling (e.g., 10×2, 10×4, 6×4, 10x4) yield higher effect sizes, indicating that predictive guidance benefits from more informative search spaces. TransFuser proved more failure-prone, and thus simpler configurations (e.g., 10×1, 10×2) are sufficient.

For TransFuser, configurations with clip duration=6s produce a negligible number of collisions with any technique, indicating this configuration is not ideal for testing TransFuser. For configurations with clip duration=10s, Foresee has an advantage over SelfOracle with medium effect sizes for configurations 10x1 and 10x2, as well as a small effect size for 10x4. However, Foresee has only a small effect size over Random in configuration 10x1, 10x2, with a very small effect size (Cohen's d=0.11) in configuration 10x4.

We provide detailed comparisons for the configuration 10x4, which is the one with the best failure rate from RQ$_1$ for InterFuser. For TransFuser, failure rates across all configurations with clip duration=10 are very similar, so we chose 10x4 for consistency. Table 3 shows, for each technique (rows) and SUT (InterFuser above, TransFuser below), the average number of failures and their nature. Column #TCs shows the number of test cases associated with a given scenario. Columns F1, F2, F3, F4, and F5 show the different kinds of collisions detected, respectively related to collisions involving the ego vehicle with elements beyond the road, such as pavements or poles (F1), pedestrians (F2), and frontal (F3), lateral (F4), or rear (F5) collisions with other vehicles. Column Σ shows totals. Results are presented for each scenario separately, as well as an aggregate. The table reinforces the effectiveness of Foresee over the baselines, across all scenarios, failure types, and SUTs. Overall, Foresee achieves, for InterFuser, a failure rate increase of +186.69% and +44.09% with respect to Random and SelfOracle.

For TransFuser, the failure rate increases were +37.06% and +21.38% over Random and SelfOracle, respectively. Over the two SUTs, Foresee achieves +128.70% and +38.09% failure rate compared to Random and SelfOracle. It also achieves a higher diversity of failure kinds observed, even though failure F4 (lateral collisions with other vehicles) is the most prevalent.

To assess the contribution of Foresee's full risk-based ranking strategy, Table 5 reports the results for both the Proximity-based

Table 5: RQ$_2$: Effectiveness of Foresee risk-based ranking: breakdown of collisions per scenario and configuration. #RPs=Number of risky points. #C. = Number of collisions.

	Proximity rank						**Risk-based rank**					
n_{rp}	1		2		4		1		2		4	
	#RPs	#C.	#RPs	#C.	#RPs	#C.	#RPs	#C.	#RPs	#C.	#RPs	#C.
InterFuser												
Scenario3	15	0	29	1	58	7	15	6	30	8	59	13
Scenario4	37	1	73	6	131	8	35	11	66	26	95	39
Scenario7	11	2	23	3	40	3	11	1	25	7	41	22
Scenario8	17	2	33	4	62	7	13	2	28	12	42	14
Scenario9	10	4	23	5	44	5	14	4	25	4	37	5
Σ	90	9	181	19	335	30	88	24	174	57	274	93
#C / #RP	0.10		0.10		0.08		0.27		0.32		**0.33**	
TransFuser												
Scenario3	15	5	29	8	55	13	15	7	29	11	51	17
Scenario4	39	2	76	2	141	3	29	4	47	9	64	9
Scenario7	15	0	29	1	57	6	12	2	16	2	18	2
Scenario8	18	2	36	3	67	7	12	0	17	0	21	0
Scenario9	17	3	33	3	64	5	13	0	23	0	25	0
Σ	104	12	203	17	384	34	81	13	132	22	179	28
#C / #RP	0.12		0.08		0.09		0.16		**0.17**		0.16	

NPC selection baseline (Proximity rank, left-hand side) and Foresee's complete ranking method (Foresee, right-hand side). The table shows the number of risky points evaluated and the number of collisions induced across scenarios, for different #RP values.

Across both systems under test, Foresee's ranking strategy consistently improves efficiency, measured as collisions per risky point, compared to the proximity-only baseline. Under InterFuser, Foresee produces both more total collisions and higher efficiency. For example, with #RP = 2, Foresee triggers 57 collisions from 174 risky points (0.33 collisions per RP), while the proximity baseline triggers just 19 from 181 points (0.10 per RP). With #RP = 4, the difference is even more pronounced: 93 collisions for Foresee (0.34 per RP) versus 30 for proximity (0.09 per RP). On Transfuser, the trend is more nuanced. The proximity baseline reaches slightly more total collisions (e.g., 34 vs. 28 at #RP = 4), but it requires significantly more exploration, 384 risky points compared to 179. This results in lower efficiency (0.09 collisions per RP vs. 0.16 for Foresee). Thus, while the proximity strategy occasionally triggers more failures in absolute terms, Foresee achieves comparable or better outcomes with far fewer test executions, making it more cost-effective.

RQ$_2$ (comparison): *Foresee outperforms the considered baselines, with a failure rate increase of +128.70% and +38.09% with respect to Random and SelfOracle, respectively. Ablation results confirm that Foresee risk-based ranking improves failure discovery efficiency over a proximity-only baseline, across both systems under test, highlighting the benefit of Foresee risk-based prioritization.*

4.5.3 *RQ$_3$ (efficiency).*

The Exhaustive search, executed on InterFuser, showed the maximum number of collisions discoverable, but Foresee is more efficient in finding the collisions. For $o_b + o_a = 6s$ and $n_{rp} = 4$, Exhaustive exposed 4.2 collisions per hour (271/64h) compared to 15.53 collisions per hour (73/4.7h) by Foresee, resulting in a 269.76% efficiency increase. Similarly, for $o_b + o_a = 10s$ and $n_{rp} = 4$, Exhaustive discovered 3.5 collisions per hour (233.33/66.5h) compared to 14.64 collisions per hour (93.67/6.4h) discovered by Foresee, a 318.29% increase.

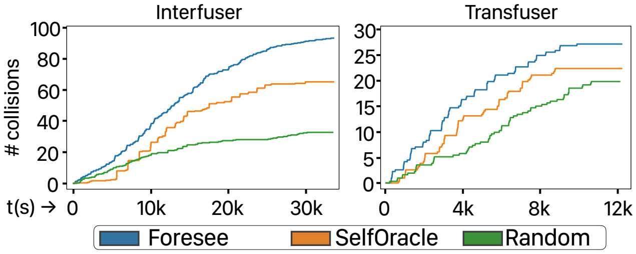

Figure 4: RQ$_3$: Efficiency of Random, SelfOracle, and Foresee for configuration 10 x 4 ($o_b + o_a = 10$ and $n_{rp} = 4$).

Figure 4 shows, for both InterFuser and TransFuser, the cumulative number of collisions detected by the techniques over time and the area under the curve (AUC) associated with the corresponding plots. The result indicates the superior ability of Foresee over the baselines to efficiently expose failures, as evidenced by the position of Foresee's plot relative to the plot of the other techniques (and higher AUC score). Foresee has an AUC score of 1910171.65, which is 1.44×higher than the AUC score of SelfOracle and 2.62×higher than the AUC score of Random for InterFuser, and shows a similar trend for Transfuser (AUC=223400.39, an increase of 1.27× and 1.74× against SelfOracle and Random). Across both SUTs, Foresee is 1.42× faster than SelfOracle and 2.49× faster than Random.

RQ$_3$ (efficiency): *Foresee identifies collisions significantly faster than the baselines. For InterFuser, the AUC of Foresee is 1.44× higher than SelfOracle's and 2.62× higher than Random's, while for Transfuser, AUC increases by 1.27× and 1.74× compared to SelfOracle and Random (1.42× and 2.49× faster overall).*

4.5.4 *RQ$_4$: Complementarity.*

Table 6 presents the number of infractions detected in the evaluated scenarios over two iterations of DriveFuzz, together with the additional failures identified by Foresee when applied to nonfailing cases. Each row reports the plain infractions (on the original unfuzzed routes), followed by the results of DriveFuzz and Foresee, for each iteration. The data illustrate how Foresee uncovers additional failures beyond those detected by DriveFuzz alone.

For InterFuser, in Iteration 1, DriveFuzz discovered 18 infractions, while Foresee found 19 additional ones. In Iteration 2, DriveFuzz added 15 more, and Foresee contributed 12 additional failures. This increased the total failure count from 17 (baseline) to 81, corresponding to a 376.47% increase. Foresee alone uncovered 31 failures, compared to 33 by DriveFuzz, which is a 93.94% increase over DriveFuzz's findings.

For Transfuser, DriveFuzz uncovered 21 infractions in Iteration 1 and 16 in Iteration 2, while Foresee identified 11 and 6 additional failures, respectively. The cumulative failure count thus rose from 9 to 63, a 600.00% increase over the baseline. Foresee contributed 17 failures in total, an increase of 45.95% compared to the 37 found by DriveFuzz. Notably, in Scenarios 7 and 8 for Transfuser, which had zero baseline infractions, Drivefuzz and Foresee combined revealed multiple failures (10 and 9).

Table 6: RQ$_4$: Effectiveness of Foresee with DriveFuzz: failures discovered across iterations. #F=Number of failures. +%=Percentage increase w.r.t. Plain.

		Iteration 1				Iteration 2			
	Plain	DriveFuzz	Foresee	Σ		DriveFuzz	Foresee	Σ	
	#F	+#F	+#F	#F	+%	+#F	+#F	#F	+%
InterFuser									
Scenario 3	1	3	2	6	500	2	1	9	800
Scenario 4	6	4	5	15	150	7	5	27	350
Scenario 7	5	3	3	11	120	0	1	12	140
Scenario 8	2	6	7	15	650	3	5	23	1050
Scenario 9	3	2	2	7	133	3	0	10	233
Σ	17	18	19	54	217.65	15	12	81	376.47
TransFuser									
Scenario 3	1	4	4	8	700	8	1	18	1700
Scenario 4	5	9	1	10	100	5	1	21	320
Scenario 7	0	4	2	6	-	0	4	10	-
Scenario 8	0	3	4	7	-	2	0	9	-
Scenario 9	3	1	0	4	33	1	0	5	66.67
Σ	9	21	11	41	355.56	16	6	63	600

RQ$_4$ (complementarity): *Foresee complements the state-of-the-art fuzzer DriveFuzz by uncovering 31 additional failures for InterFuser and 17 for TransFuser, corresponding to a 93.94% and 45.95% increase over the failures found by DriveFuzz alone. In multiple scenarios (e.g., Scenario 7 and 8 in TransFuser), Foresee detected failures where DriveFuzz found none, suggesting its effectiveness in extending the coverage of existing fuzzing campaigns.*

4.5.5 *RQ$_5$: Generalizability.* Table 7a shows Foresee's capability of revealing infractions on four additional maps: Town01, Town02, To,wn03 and Town04 with the InterFuser and Transfuser agents. Foresee can effectively discover infractions of different categories across all towns and agents. We find that the most common infraction type is F4 (lateral collision), observed 181 t,imes whereas F3 (frontal collision) is the most uncommon, observed 10 times. For Transfuser, we find 115 previously unknown collisions across 1,079 test cases. For InterFuser, we find 176 previously unknown collisions across 1,307 test cases.

Table 7b show Foresee's effectiveness with the Apollo [3] framework. Unlike InterFuser and Transfuser which are end-to-end systems, Apollo is a multi-modal system [60] where different modules (e.g., planner, control, PID) control different aspects of driving. Consequently, its behavior differs from that of the other agents, as reflected in the distribution of collision types in Table 7b. Aside from a few lateral collisions and instances where the vehicle remained stationary, most collisions were classified as type F1 (object). We discuss these failures in more detail in Section 4.6.

Table 7c demonstrates Foresee's ability to reveal infractions under heavy weather conditions. In these conditions, both Transfuser and InterFuser reveal 34 lateral collisions (F4) which is the majority. It also reveals 8 pedestrian (F2) and 4 read (F4) collisions and avoids frontal/object collisions. In heavy weather, both agents drive more carefully and slowly to avoid frontal collisions and colliding with roadside collisions, but the effects of low visibility causes lateral and pedestrian collisions, while driving too slow in risky situations causes some rear collisions.

Table 7: RQ$_5$: Generalizability. Impact of town, agent (Apollo), and heavy weather on infractions per category. Configuration 10 x 4 ($o_b + o_a = 10, n_{rp} = 4$).

(a) Towns

Map	Agent	#TCs	F1	F2	F3	F4	F5	Stuck	Σ
Town01	InterFuser	238	8	3	0	17	4	0	32
	Transfuser	205	0	0	0	10	6	0	16
Town02	InterFuser	160	22	0	5	17	3	0	47
	Transfuser	143	0	0	0	1	0	0	1
Town03	InterFuser	512	0	0	1	25	0	0	27
	Transfuser	324	0	15	0	15	0	0	30
Town04	InterFuser	397	24	0	0	44	1	0	70
	Transfuser	407	0	0	4	52	12	0	68
Σ		2386	54	18	10	181	26	0	291

(b) Apollo

Map	#TCs	F1	F2	F3	F4	F5	Stuck	Σ
Town01	80	15	0	0	1	0	0	16
Town03	64	6	0	0	3	0	5	14
Σ	144	21	0	0	4	0	5	30

(c) Heavy weather

Agent	#TCs	F1	F2	F3	F4	F5	Stuck	Σ
InterFuser	539	0	3	0	29	4	0	36
Transfuser	237	0	5	0	5	0	0	10
Σ	776	0	8	0	34	4	0	46

RQ$_5$ (generalizability): *Foresee is capable of generalizing under different maps, weather conditions, and an industrial-grade multimodal agent, Apollo. For example, Foresee reveals 291 collisions from previously failure free scenarios in four different towns with Transfuser and InterFuser, 30 new collisions with Apollo, and 46 new collisions under heavy weather condition.*

4.6 Qualitative Analysis

To evaluate whether Foresee exploits segments already close to failure or uncovers meaningful, non-trivial vulnerabilities, we conducted a qualitative analysis on the data from the exhaustive search (RQ$_1$). For each failure-free test, we found the most critical ego–NPC interactions using simulation logs and characterized them using three interpretable features: minimum time-to-collision (TTC), relative distance, and relative speed. We labeled each interaction based on whether it led to a failure during fuzzing and trained a Random Forest classifier [19] on each scenario to predict failure-prone interactions. The models achieved recall in the 0.64-0.86 range (except for Scenario 9 at 0.31), and precision in the 0.12-0.41 range. Results suggest that, while failures can often be detected, the boundary between safe and unsafe behaviors is complex.

Failures rarely stemmed from a single extreme factor (e.g., low TTC), but instead from nuanced combinations. In Scenario 3, failures occurred with high TTC (>1482s) and long distances (>150m) when paired with modest speeds (>−3.1m/s). Scenario 4 showed failures at distances up to 83.99m when relative speed was mild (>−4.2m/s). In Scenario 8, short distances (≤3.29m) combined with low relative speeds (>0.0m/s) triggered failures. Scenario 9 revealed failures even with high TTC (>30s).

Regarding Apollo's failures, the driving agent generally performs well in avoiding other vehicles under nominal conditions. However, when a nearby NPC vehicle encroaches on the ego lane, the local planner tends to generate sharper avoidance trajectories. This behavior leads the control module to produce oscillatory, wobble-like movements, resembling the response of a PID controller under suboptimal tuning. These oscillations can amplify over time, causing the vehicle to drift off the lane and eventually collide with static objects such as walls, poles, or benches—obstacles that the perception system does not consistently detect in advance. Consequently, most failures correspond to object collisions (type F1). Under normal driving conditions, these crashes do not occur; the revealed failures highlight a limited ability of Apollo's planning and control modules to generalize to such corner cases.

4.7 Threats to Validity

Internal Validity. All variants of Foresee, SelfOracle, and random were evaluated under identical experimental conditions and the same test set. The main threat concerns the correctness of our test script implementations, which we verified thoroughly. We could not train SelfOracle using the same dataset as InterFuser, nor use pre-trained models due to simulator differences. We mitigated this by training the VAE following standard guidelines [50, 51], and our version of SelfOracle performs competitively.

Regarding the ADS, which could bias results if unsuitable for driving, we used two top-ranking, publicly available models from the CARLA leaderboard. As for the simulation platform, we adopted CARLA, widely used in recent failure prediction research [57].

One threat relates to the quality of input test cases, specifically, whether scenarios already include near-misses, making failures easier to induce. To assess this, we analyzed whether failures found by Foresee stemmed from obviously risky inputs. As discussed in Section 4.6, decision tree analyses show that failures arise from subtle, multi-feature conditions rather than extreme TTC, distance, or speed values, suggesting Foresee reveals non-trivial vulnerabilities.

Another concern involves the realism of scenario mutations. Unlike global mutation tools like DriveFuzz [28], our method applies minimal, targeted modifications to existing simulations, preserving NPC count and adjusting only those closest to the ego vehicle. We use conservative perturbations, such as slight steering changes or vehicle swaps within the same class, aligned with localization error thresholds [41] and established targeted testing techniques [7]. Limited repetition of our experiments may affect the robustness of our results, as simulation-based testing is computationally expensive.

External Validity. The limited number of self-driving systems in our evaluation constitutes a threat to the generalizability of our results to other ADS. Results may not generalize when considering other simulation platforms or ADSs.

Reproducibility. All software artifacts and the complete results set are available in our replication package [56].

5 RELATED WORK

Test Generation for Autonomous Driving. The majority of test generation techniques employ search-based techniques for DNN-based ADS testing [1, 4, 5, 27, 36, 40, 42, 55, 63]. Mullins et al. [37] use Gaussian processes to drive the search-based test generation, whereas Gambi et al. [15] propose search-based test generation for ADS based on procedural content generation. DriveFuzz uses the physical states of the vehicle and oracles based on real-world traffic rules to guide the fuzzer towards finding misbehaviors [28]. AutoFuzz [64] focuses on fuzzing the test scenario specification. Before fuzzing, it uses a seed selection mechanism based on a binary classifier that selects likely traffic-violating seeds. AV-Fuzzer [31] uses a genetic algorithm that is informed by the positioning of globally monitored NPCs in each scenario in the driving environment. The NPCs with higher potential of safety potential [22] violation likelihood are selected for evolution. Cheng et al. [9] propose BehaviorMiner, an unsupervised model that extracts the temporal features from certain given scenarios and performs a clustering-based abstraction to group behaviors with similar features into abstract states. Neelofar and Aleti [38] propose characterizing critical scenarios for ADS using a combination of static and dynamic features. SimADFuzz [61] uses distance-guided mutation strategies to enhance the probability of interactions among vehicles in generated scenarios. Koren et al. [30] and Corso et al. [12] use Monte Carlo tree search and deep reinforcement learning to find collision scenarios. Similarly, GARL [32] uses reinforcement learning to detect violations in marker-based autonomous landing systems.

Test generators are designed to maximize the number of failures and consider *whole* test cases. While the exploration is guided towards critical regions, the search budget is consumed by running test cases that do not result in failing conditions. Our approach differentiates itself by forecasting potential ego vehicle states to predict infractions with NPCs and focuses on local segments within test cases. Existing fuzzers/test generators do not constitute direct baselines for our approach, as they operate at the beginning of each scenario by creating new test scenarios at each iteration to maximize failure exposure. Thus, no simulation effort is reused as they operate globally, other than locally. However, they can be used in conjunction with Foresee to improve their effectiveness, especially for the false negative cases, as shown with DriveFuzz.

Focused Test Generation for Autonomous Driving. Although focused test case generation has been a subject of extensive study and application in the context of software testing, its application to ADS is a new field that has been underexplored. To the best of our knowledge, only two approaches have been proposed. DeepAtash-LR by Zohdinasab et al. [66] improves targeted test generation by using a surrogate model to avoid executing whole test cases. In contrast, we focus on reusing simulation segments. TM-Fuzzer [33] dynamically generates NPCs in the neighborhood of the ego vehicle to increase the likelihood of failures. Differently, we keep the number of NPCs the same and introduce only a minimal modification by changing their type, which impacts their kinematic behavior. Also, our framework promotes the reuse of simulation resources, which is not the case with any of the existing approaches.

Anomaly Detection in Autonomous Driving. We already discussed SelfOracle [51], for which we performed an explicit empirical comparison in this work. Similarly, DeepGuard [20] uses the reconstruction error by VAEs to prevent collisions of vehicles with the roadside. ThirdEye [47] uses the attention maps from the explainable AI domain to predict misbehaviors of self-driving cars. DeepRoad [63] validates single driving images based on the distance to the training set, using embeddings rooted in the features extracted by VGGNet [46]. Other works [49, 50] use continual learning to minimize the false positives of a black-box failure predictor, whereas other researchers use uncertainty quantification [16] or probabilistic time-series [45].

Our approach differs from the aforementioned approaches because it uses a risky score of the system synthesized from a forecasting mechanism of the ego vehicle kinetics.

6 CONCLUSIONS

Simulation-based testing is highly useful in autonomous vehicle testing, but the cost of revealing faults relative to the time to run simulations is very high. We propose Foresee, an approach to optimize simulation-based testing by reusing segments of simulations that produce near-failing situations. Our approach uses a custom misbehavior forecaster to detect near misses and fuzz the state of the simulation locally to produce failures quickly. Experimental results show that failure-free scenarios embed many near-failing situations that Foresee can accurately detect, and that many of these cases result in failures when minimal perturbations are introduced. Foresee provides initial yet strong evidence that guiding fuzzing with misbehavior forecasting is a promising approach to uncovering hidden failures in ADS. In the future, we plan to evaluate additional forecasting methods based on multi-horizon and multi-variate time series.

REFERENCES

[1] Raja Ben Abdessalem, Annibale Panichella, Shiva Nejati, Lionel C. Briand, and Thomas Stifter. 2018. Testing Autonomous Cars for Feature Interaction Failures Using Many-objective Search. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE 2018)*. ACM.

[2] Hervé Abdi. 2010. Holm's sequential Bonferroni procedure. *Encyclopedia of research design* 1, 8 (2010), 1–8.

[3] Baidu. 2021. Apollo ADS platform. https://github.com/ApolloAuto/apollo

[4] R. Ben Abdessalem, S. Nejati, L. C. Briand, and T. Stifter. 2016. Testing advanced driver assistance systems using multi-objective search and neural networks. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*.

[5] R. Ben Abdessalem, S. Nejati, L. C. Briand, and T. Stifter. 2018. Testing Vision-Based Control Systems Using Learnable Evolutionary Algorithms. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*.

[6] Matteo Biagiola, Andrea Stocco, Vincenzo Riccio, and Paolo Tonella. 2024. Two is Better Than One: Digital Siblings to Improve Autonomous Driving Testing. *Empirical Software Engineering* (2024).

[7] Matteo Biagiola and Paolo Tonella. 2024. Boundary State Generation for Testing and Improvement of Autonomous Driving Systems. *IEEE Transactions on Software Engineering* 50, 8 (2024), 2040–2053. https://doi.org/10.1109/TSE.2024.3420816

[8] Georg Burkhard, S. Vos, N. Munzinger, E. Enders, and D. Schramm. 2018. Requirements on driving dynamics in autonomous driving with regard to motion and comfort. In *18. Internationales Stuttgarter Symposium*. Springer Fachmedien Wiesbaden.

[9] Mingfei Cheng, Yuan Zhou, and Xiaofei Xie. 2023. BehAVExplor: Behavior Diversity Guided Testing for Autonomous Driving Systems. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis* (Seattle, WA, USA) *(ISSTA 2023)*. Association for Computing Machinery, New York, NY, USA, 488–500. https://doi.org/10.1145/3597926.3598072

[10] Kashyap Chitta, Aditya Prakash, Bernhard Jaeger, Zehao Yu, Katrin Renz, and Andreas Geiger. 2022. TransFuser: Imitation with Transformer-Based Sensor Fusion for Autonomous Driving. *Pattern Analysis and Machine Intelligence (PAMI)* (2022).

[11] Jacob Cohen. 1988. *Statistical power analysis for the behavioral sciences*. L. Erlbaum Associates, Hillsdale, N.J.

[12] Anthony Corso, Peter Du, Katherine Driggs-Campbell, and Mykel J. Kochenderfer. 2019. Adaptive Stress Testing with Reward Augmentation for Autonomous Vehicle Validatio. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)* (Auckland, New Zealand). IEEE Press, 163–168. https://doi.org/10.1109/ITSC.2019.8917242

[13] Alexey Dosovitskiy, Germán Ros, Felipe Codevilla, Antonio López, and Vladlen Koltun. 2017. CARLA: An Open Urban Driving Simulator. *CoRR* abs/1711.03938 (2017).

[14] EpicGames. 2023. UnrealEngine4. https://www.unrealengine.com.

[15] Alessio Gambi, Marc Mueller, and Gordon Fraser. 2019. Automatically Testing Self-driving Cars with Search-based Procedural Content Generation. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2019)*. ACM.

[16] Ruben Grewal, Paolo Tonella, and Andrea Stocco. 2024. Predicting Safety Misbehaviours in Autonomous Driving Systems using Uncertainty Quantification. In *Proceedings of 17th IEEE International Conference on Software Testing, Verification and Validation (ICST '24)*. IEEE, 12 pages.

[17] Fitash Ul Haq, Donghwan Shin, Shiva Nejati, and Lionel Briand. 2020. Comparing Offline and Online Testing of Deep Neural Networks: An Autonomous Car Case Study. In *Proceedings of 13th IEEE International Conference on Software Testing, Verification and Validation (ICST '20)*. IEEE.

[18] Fitash Ul Haq, Donghwan Shin, Shiva Nejati, and Lionel Briand. 2021. Can Offline Testing of Deep Neural Networks Replace Their Online Testing? A Case Study of Automated Driving Systems. *Empirical Softw. Engg.* 26, 5 (jul 2021), 30 pages. https://doi.org/10.1007/s10664-021-09982-4

[19] Tin Kam Ho. 1995. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, Vol. 1. IEEE, 278–282.

[20] Manzoor Hussain, Nazakat Ali, and Jang-Eui Hong. 2022. DeepGuard: A Framework for Safeguarding Autonomous Driving Systems from Inconsistent Behaviour. *Automated Software Engg.* 29, 1 (may 2022), 32 pages. https://doi.org/10.1007/s10515-021-00310-0

[21] Bernhard Jaeger, Kashyap Chitta, and Andreas Geiger. 2023. Hidden Biases of End-to-End Driving Models. *arXiv preprint arXiv:2306.07957* (2023).

[22] Saurabh Jha, Subho Sankar Banerjee, Timothy Tsai, Siva Kumar Sastry Hari, Michael B. Sullivan, Zbigniew T. Kalbarczyk, Stephen W. Keckler, and Ravishankar Krishnan Iyer. 2019. ML-Based Fault Injection for Autonomous Vehicles: A Case for Bayesian Fault Injection. *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (2019), 112–124. https://api.semanticscholar.org/CorpusID:195776612

[23] Xiaosong Jia, Yulu Gao, Li Chen, Junchi Yan, Patrick Langechuan Liu, and Hongyang Li. 2023. Driveadapter: Breaking the coupling barrier of perception and planning in end-to-end autonomous driving. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 7953–7963.

[24] Xiaosong Jia, Penghao Wu, Li Chen, Jiangwei Xie, Conghui He, Junchi Yan, and Hongyang Li. 2023. Think Twice before Driving: Towards Scalable Decoders for End-to-End Autonomous Driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 21983–21994.

[25] Shinpei Kato, Shota Tokunaga, Yuya Maruyama, Seiya Maeda, Manato Hirabayashi, Yuki Kitsukawa, Abraham Monrroy, Tomohito Ando, Yusuke Fujii, and Takuya Azumi. 2018. Autoware on Board: Enabling Autonomous Vehicles with Embedded Systems. In *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*. 287–296. https://doi.org/10.1109/ICCPS.2018.00035

[26] Prabhjot Kaur, Samira Taghavi, Zhaofeng Tian, and Weisong Shi. 2021. A Survey on Simulators for Testing Self-Driving Cars. arXiv:2101.05337 [cs.RO]

[27] Jinhan Kim, Robert Feldt, and Shin Yoo. 2019. Guiding Deep Learning System Testing Using Surprise Adequacy. In *Proceedings of the 41st International Conference on Software Engineering (ICSE '19)*. IEEE Press.

[28] Seulbae Kim, Major Liu, Junghwan "John" Rhee, Yuseok Jeon, Yonghwi Kwon, and Chung Hwan Kim. 2022. DriveFuzz. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. ACM. https://doi.org/10.1145/3548606.3560558

[29] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014). https://api.semanticscholar.org/CorpusID:6628106

[30] Mark Koren, Saud Alsaif, Ritchie Lee, and Mykel J. Kochenderfer. 2018. Adaptive Stress Testing for Autonomous Vehicles. In *2018 IEEE Intelligent Vehicles Symposium (IV)*. 1–7. https://doi.org/10.1109/IVS.2018.8500400

[31] Guanpeng Li, Yiran Li, Saurabh Jha, Timothy Tsai, Michael Sullivan, Siva Kumar Sastry Hari, Zbigniew Kalbarczyk, and Ravishankar Iyer. 2020. AV-FUZZER: Finding Safety Violations in Autonomous Driving Systems. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. 25–36. https://doi.org/10.1109/ISSRE5003.2020.00012

[32] Linfeng Liang, Yao Deng, Kye Morton, Valtteri Kallinen, Alice James, Avishkar Seth, Endrowednes Kuantama, Subhas Mukhopadhyay, Richard Han, and Xi

Zheng. 2025. GARL: Genetic Algorithm-Augmented Reinforcement Learning to Detect Violations in Marker-Based Autonomous Landing Systems. In *Proceedings of the IEEE/ACM 47th International Conference on Software Engineering* (Ottawa, Ontario, Canada) *(ICSE '25)*. IEEE Press, 411–423. https://doi.org/10.1109/ICSE55347.2025.00076

[33] Shenghao Lin, Fansong Chen, Laile Xi, Gaosheng Wang, Rongrong Xi, Yuyan Sun, and Hongsong Zhu. 2024. TM-fuzzer: fuzzing autonomous driving systems through traffic management. *Automated Software Engineering* 31, 2 (2024), 61.

[34] Guannan Lou, Yao Deng, Xi Zheng, Mengshi Zhang, and Tianyi Zhang. 2022. Testing of autonomous driving systems: where are we and where should we go?. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Singapore, Singapore) *(ESEC/FSE 2022)*. Association for Computing Machinery, New York, NY, USA, 31–43. https://doi.org/10.1145/3540250.3549111

[35] Chengjie Lu, Yize Shi, Huihui Zhang, Man Zhang, Tiexin Wang, Tao Yue, and Shaukat Ali. 2023. Learning Configurations of Operating Environment of Autonomous Vehicles to Maximize their Collisions. *IEEE Transactions on Software Engineering* 49, 1 (2023), 384–402. https://doi.org/10.1109/TSE.2022.3150788

[36] Mahshid Helali Moghadam, Markus Borg, Mehrdad Saadatmand, Seyed Jalaleddin Mousavirad, Markus Bohlin, and Björn Lisper. 2024. Machine learning testing in an ADAS case study using simulation-integrated bio-inspired search-based testing. *J. Softw. Evol. Process* 36, 5 (April 2024), 25 pages. https://doi.org/10.1002/smr.2591

[37] Galen E. Mullins, Paul G. Stankiewicz, R. Chad Hawthorne, and Satyandra K. Gupta. 2018. Adaptive generation of challenging scenarios for testing and evaluation of autonomous vehicles. *Journal of Systems and Software* 137 (2018).

[38] Neelofar Neelofar and Aldeida Aleti. 2024. Identifying and Explaining Safety-critical Scenarios for Autonomous Vehicles via Key Features. *ACM Trans. Softw. Eng. Methodol.* 33, 4, Article 94 (April 2024), 32 pages. https://doi.org/10.1145/3640335

[39] U.S. Department of Transportation. 2017. Traffic Safety Facts: Research Notes. https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812456.

[40] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. DeepXplore: Automated Whitebox Testing of Deep Learning Systems. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP '17)*. ACM.

[41] Tyler G. R. Reid, Sarah E. Houts, Robert Cammarata, Graham Mills, Siddharth Agarwal, Ankit Vora, and Gaurav Pandey. 2019. Localization Requirements for Autonomous Vehicles. *CoRR* abs/1906.01061 (2019). arXiv:1906.01061 http://arxiv.org/abs/1906.01061

[42] Vincenzo Riccio and Paolo Tonella. 2020. Model-Based Exploration of the Frontier of Behaviours for Deep Learning System Testing. In *Proceedings of ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '20)*.

[43] Guodong Rong, Byung Hyun Shin, Hadi Tabatabaee, Qiang Lu, Steve Lemke, Mārtiņš Možeiko, Eric Boise, Geehoon Uhm, Mark Gerow, Shalin Mehta, et al. 2020. LGSVL Simulator: A High Fidelity Simulator for Autonomous Driving. *arXiv preprint arXiv:2005.03778* (2020).

[44] Hao Shao, Letian Wang, Ruobing Chen, Hongsheng Li, and Yu Liu. 2023. Safety-enhanced autonomous driving using interpretable sensor fusion transformer. In *Conference on Robot Learning*. PMLR, 726–737.

[45] Sepehr Sharifi, Andrea Stocco, and Lionel C. Briand. 2025. System Safety Monitoring of Learned Components Using Temporal Metric Forecasting. *ACM Transactions on Software Engineering and Methodology* (jan 2025). https://doi.org/10.1145/3712196

[46] Karen Simonyan and Andrew Zisserman. [n.d.]. Very Deep Convolutional Networks for Large-Scale Image Recognition. ([n. d.]). arXiv:1409.1556v6 [cs.CV]

[47] Andrea Stocco, Paulo J. Nunes, Marcelo d'Amorim, and Paolo Tonella. 2022. ThirdEye: Attention Maps for Safe Autonomous Driving Systems. In *Proceedings of 37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22)*. IEEE/ACM. https://doi.org/10.1145/3551349.3556968

[48] Andrea Stocco, Brian Pulfer, and Paolo Tonella. 2022. Mind the Gap! A Study on the Transferability of Virtual vs Physical-world Testing of Autonomous Driving Systems. *IEEE Transactions on Software Engineering* (2022). https://doi.org/10.1109/TSE.2022.3202311

[49] Andrea Stocco and Paolo Tonella. 2020. Towards Anomaly Detectors that Learn Continuously. In *Proceedings of 31st International Symposium on Software Reliability Engineering Workshops (ISSREW 2020)*. IEEE.

[50] Andrea Stocco and Paolo Tonella. 2021. Confidence-driven Weighted Retraining for Predicting Safety-Critical Failures in Autonomous Driving Systems. *Journal of Software: Evolution and Process* (2021). https://doi.org/10.1002/smr.2386

[51] Andrea Stocco, Michael Weiss, Marco Calzana, and Paolo Tonella. 2020. Misbehaviour Prediction for Autonomous Driving Systems. In *Proceedings of 42nd International Conference on Software Engineering (ICSE '20)*. ACM.

[52] Shuncheng Tang, Zhenya Zhang, Yi Zhang, Jixiang Zhou, Yan Guo, Shuang Liu, Shengjian Guo, Yan-Fu Li, Lei Ma, Yinxing Xue, and Yang Liu. 2022. A Survey on Automated Driving System Testing: Landscapes and Trends. https://doi.org/10.48550/ARXIV.2206.05961

[53] CARLA team. 2023. CARLA: Leaderboard. https://leaderboard.carla.org/leaderboard/

[54] CARLA team. 2023. CARLA: Traffic Scenarios. https://leaderboard.carla.org/scenarios/

[55] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. DeepTest: Automated Testing of Deep-neural-network-driven Autonomous Cars. In *Proceedings of the 40th International Conference on Software Engineering (ICSE '18)*. ACM.

[56] Foresee 2025. Replication Package. https://www.dropbox.com/scl/fo/soc3eur1d4jcsa8mlbxpu/ANuBsDGb6EqKQH734ZUhzlQ?rlkey=palf178kd5g3uxshv0yn4lxw1&st=rkt2vlbp&dl=0.

[57] Fitash Ul Haq, Donghwan Shin, and Lionel C. Briand. 2023. Many-Objective Reinforcement Learning for Online Testing of DNN-Enabled Systems. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. 1814–1826. https://doi.org/10.1109/ICSE48619.2023.00155

[58] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[59] Frank Wilcoxon. 1945. Individual Comparisons by Ranking Methods. *Biometrics Bulletin* 1, 6 (Dec. 1945), 80. https://doi.org/10.2307/3001968

[60] Yi Xiao, Felipe Codevilla, Akhil Gurram, Onay Urfalioglu, and Antonio M López. 2020. Multimodal end-to-end autonomous driving. *IEEE Transactions on Intelligent Transportation Systems* 23, 1 (2020), 537–547.

[61] Huiwen Yang, Yu Zhou, and Taolue Chen. 2024. SimADFuzz: Simulation-Feedback Fuzz Testing for Autonomous Driving Systems. arXiv:2412.13802 [cs.SE] https://arxiv.org/abs/2412.13802

[62] Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. 2020. A survey of autonomous driving: Common practices and emerging technologies. *IEEE access* 8 (2020), 58443–58469.

[63] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. 2018. DeepRoad: GAN-based Metamorphic Testing and Input Validation Framework for Autonomous Driving Systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE 2018)*. ACM.

[64] Ziyuan Zhong, Gail Kaiser, and Baishakhi Ray. 2021. Neural Network Guided Evolutionary Fuzzing for Finding Traffic Violations of Autonomous Vehicles.

[65] Ziyuan Zhong, Yun Tang, Yuan Zhou, Vania de Oliveira Neves, Yang Liu, and Baishakhi Ray. 2021. A Survey on Scenario-Based Testing for Automated Driving Systems in High-Fidelity Simulation. arXiv:2112.00964 [cs.SE] https://arxiv.org/abs/2112.00964

[66] Tahereh Zohdinasab, Vincenzo Riccio, and Paolo Tonella. 2024. Focused Test Generation for Autonomous Driving Systems. *ACM Trans. Softw. Eng. Methodol.* 33, 6, Article 152 (June 2024), 32 pages. https://doi.org/10.1145/3664605